B.M.S College of Engineering

P.O. Box No.: 1908 Bull Temple Road,

Bangalore-560 019

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Course – Unix System Programming

Course Code – 19IS4PWUSP

AY 2019-20

Report on Unix System Programming Project

“A dice game to practice process synchronization in Unix.”

Submitted by

Rajath K,1BM19IS125
Rohith R Kashyap,1BM19IS131
Roopesh Reddy C,1BM19IS132

Submitted to

Chandrakala G Raju

Assistant Professor

B.M.S College of Engineering

P.O. Box No.: 1908 Bull Temple Road,

Bangalore-560 019

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

Certified that the Project has been successfully presented at **B.M.S College Of Engineering** by **Rajath K** bearing USN:**1BM19IS125**, **Roopesh Reddy C** bearing USN:**1BM19IS132** and **Rohith R Kashyap** bearing USN:**1BM19IS131** in partial fulfillment of the requirements for the IV Semester degree in **Bachelor of Engineering in Information Science & Engineering** of **Visvesvaraya Technological University, Belgaum** as a part of project for the course **Unix System Programming** Course Code- **19IS4PWUSP** during academic year 2020-2021.

Faculty Name – Chandrakala G Raju

Designation – Assistant Professor

Department of ISE, BMSCE

TABLE OF CONTENTS

ABSTRACT

Process Synchronization is a way to coordinate processes that use shared data. It occurs in an operating system among cooperating processes. ... While executing many concurrent processes, process synchronization helps to maintain shared data consistency and cooperating process execution.

This program is to illustrate how process synchronization takes place using a simple game. Three people play the dice game which acts as the child processes. These three processes take their turn of playing decided by the referee which is the parent process. The player who first gets the maximum score or points is declared as the winner.

Various APIs are used in the following program which allow access to various processes and also to give various system calls.

Introduction

It is the task phenomenon of coordinating the execution of processes in such a way that no two processes can have access to the same shared data and resources.

- It is a procedure that is involved in order to preserve the appropriate order of execution of cooperative processes.
- In order to synchronize the processes, there are various synchronization mechanisms.
- Process Synchronization is mainly needed in a multi-process system when multiple processes are running together, and more than one processes try to gain access to the same shared resource or any data at the same time.

The need for synchronization originates **when processes need to execute concurrently**. The main purpose of synchronization is the sharing of resources without interference using mutual exclusion. The other purpose is the coordination of the process interactions in an operating system.

Problem Statement

When two processes need to execute concurrently the need for process synchronization occurs.Process synchronization involves providing a time slice for each process so that they get the required time for execution.This provides the coordination of the process interactions in an operating system. Design a program which illustrates process synchronization.

WHAT ARE APIs ?

The Linux API is the kernel–user space API, which allows programs in user space to access system resources and services of the Linux kernel. It is composed out of the System Call Interface of the Linux kernel and the subroutines in the GNU C Library (glibc).

API'S used in the project

- Open()
- close()
- read()
- write()
- sleep()
- signal()
- kill()
- pause()
- lseek()

- ❖ **open() :** Used to Open the file for reading, writing or both.
  - ➢ **How it works in the os**
    - ■ Find the existing file on disk
    - ■ Create file table entry
    - ■ Set first unused file descriptor to point to file table entry
    - ■ Return file descriptor used, -1 upon failure

- **close():** Tell the operating system you are done with a file descriptor and Close the file which is pointed by fd.
  - **How it works in the OS**
    - Destroy file table entry referenced by element fd of file descriptor table – As long as no other process is pointing to it!
    - Set element fd of file descriptor table to NULL

- **read():** From the file indicated by the file descriptor fd, the read() function reads cnt bytes of input into the memory area indicated by buf. A successful read() updates the access time for the file.
  - **How it works in the OS**
    - return Number of bytes read on success
    - return 0 on reaching end of file
    - return -1 on error
    - return -1 on signal interrupt

- **write() :** Writes cnt bytes from buf to the file or socket associated with fd. cnt should not be greater than INT_MAX (defined in the limits.h header file). If cnt is zero, write() simply returns 0 without attempting any other action.
  - **How it works in the OS**
    - return Number of bytes written on success
    - return 0 on reaching end of file
    - return -1 on error
    - return -1 on signal interrupt

- **sleep() :**A computer program (process, task, or thread) may sleep, which places it into an inactive state for a period of time. Eventually the expiration of an interval timer, or the receipt of a signal or interrupt causes the program to resume execution.

- ➢ **How it works**
    - ■ A typical *sleep* system call takes a time value as a parameter, specifying the minimum amount of time that the process is to sleep before resuming execution. The parameter typically specifies seconds, although some operating systems provide finer resolution, such as ms.

❖ **signal() :** It is a mechanism by which a process may be notified of, or affected by, an event occurring in the system. The term signal is also used to refer to the event itself.

- ➢ **How it works**
    - ■ A signal is said to be delivered to a process when the specified signal-handling action for the signal is taken. A signal is said to be accepted by a process when a signal is selected and returned by one of the *sigwait* functions.

❖ **kill() :** The kill() system call can be used to send any signal to any process group or process.

- ➢ **How it works**
    - ■ If *pid* is positive, then signal *sig* is sent to *pid.*
    - ■ If *pid* equals 0, then *sig* is sent to every process in the process group of the current process.
    - ■ If *pid* equals -1, then *sig* is sent to every process for which the calling process has permission to send signals, except for process 1 (init), but see below.
    - ■ If *pid* is less than -1, then *sig* is sent to every process in the process group *-pid.*
    - ■ If *sig* is 0, then no signal is sent, but error checking is still performed.

❖ **pause() :** It causes the calling process (or thread) to sleep until a signal is delivered that either terminates the process or causes the invocation of a signal-catching function.

- **How it works**
  - returns only when a signal was caught and the signal-catching function returned.In this case, **pause**() returns -1,and *errno* is set to **EINTR.**
  - **EINTR-**a signal was caught and the signal-catching function returned.

- **lseek() :**The lseek() function shall set the file offset for the open file description associated with the file descriptor. Thus lseek allows a process to perform random access of data on any opened file.
  - **How it works**
    - Upon successful completion, the resulting offset, as measured in bytes from the beginning of the file, shall be returned. Otherwise, (**off_t**)-1 shall be returned, *errno* shall be set to indicate the error, and the file offset shall remain unchanged.

CODE

```
#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <signal.h>

#include <sys/time.h>


/* Placeholder function to avoid quitting when receiving a SIGUSR1 */

void action(){};

/* Function executed by each player, to play the dice */

void player(char *name, int playerId, int fd);

/* Function executed by the referee, to print current scores and check for a winner */

void checkWinner(int fd, char *name);

int max_size;

int main(int argc, char *argv[])

{

   int fd;

   pid_t pid1, pid2, pid3;

   printf("DiceGame: a 3-players game with a referee\n\n");


   // Creating the binary file before forking

   if ((fd = open("sharedFile.bin", O_CREAT | O_WRONLY | O_TRUNC, 0777)) == -1)

   {

      perror("File problem");

      exit(1);
```

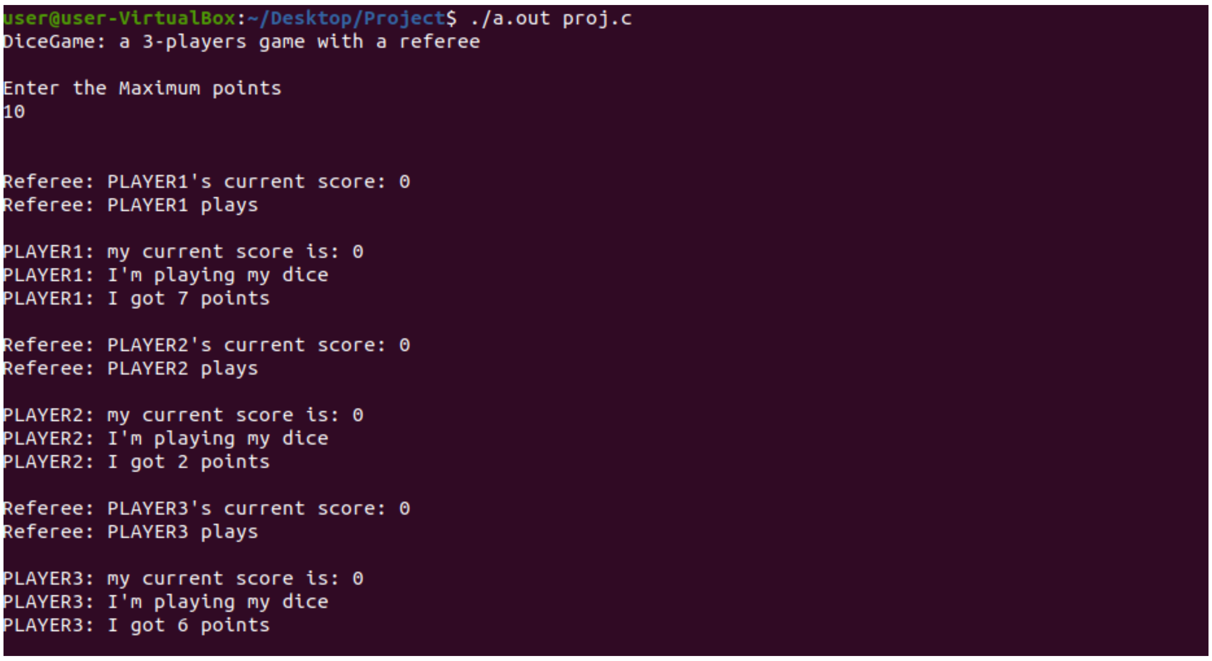
```
`

  }

   else

   {

       // Writing three zero-integer values to the file

       int *threeZeros;

       threeZeros = malloc(3 * sizeof(int));

       threeZeros[0] = 0;

       threeZeros[1] = 0;

       threeZeros[2] = 0;

       write(fd, threeZeros, 3 * sizeof(int));

       close(fd);

       free(threeZeros);

   }


printf("Enter the Maximum points\n");

scanf("%d",&max_size);

printf("\n\n");

   // Creating the players and calling the common function "player"

   if ((pid1 = fork()) == 0)

       player("PLAYER1", 1, fd);

   if ((pid2 = fork()) == 0)

       player("PLAYER2", 2, fd);

   if ((pid3 = fork()) == 0)

       player("PLAYER3", 3, fd);

   sleep(1);

   signal(SIGUSR1, action);

   while (1)

   {

       // Make the players play in order: PLAYER1, PLAYER3 then PLAYER2
```

```
            fd = open("sharedFile.bin", O_RDONLY);
        checkWinner(fd, "PLAYER1");
        printf("Referee: PLAYER1 plays\n\n");
        kill(pid1, SIGUSR1);
        pause();
        checkWinner(fd, "PLAYER2");
        printf("Referee: PLAYER2 plays\n\n");
        kill(pid2, SIGUSR1);
        pause();
        checkWinner(fd, "PLAYER3");
        printf("Referee: PLAYER3 plays\n\n");
        kill(pid3, SIGUSR1);
        pause();
    }
}

void player(char *name, int playerId, int fd)
{
    fd = open("sharedFile.bin", O_RDONLY);
    int points = 0, dice, oldScore;
    long int ss = 0;
    while (1)
    {
        signal(SIGUSR1, action);
        pause();
        // Reading the old score
        if (playerId == 1) // PLAYER1
        {
```

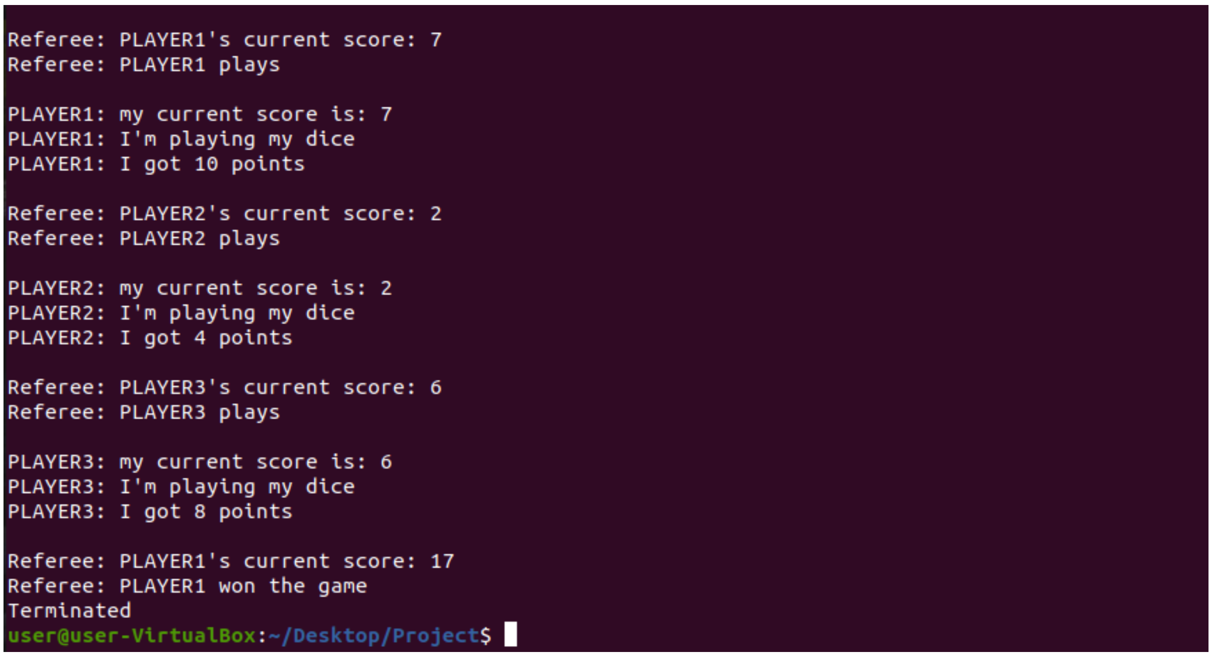
```
    lseek(fd, 0, SEEK_SET);

    read(fd, &oldScore, sizeof(int));

    printf("PLAYER1: my current score is: %d\n", oldScore);

}

else if (playerId == 2) // PLAYER3

{

    lseek(fd, sizeof(int), SEEK_SET);

    read(fd, &oldScore, sizeof(int));

    printf("PLAYER2: my current score is: %d\n", oldScore);

}

else // PLAYER2

{

    lseek(fd, 2 * sizeof(int), SEEK_SET);

    read(fd, &oldScore, sizeof(int));

    printf("PLAYER3: my current score is: %d\n", oldScore);

}

close(fd);

// Playing the dice and printing its own name and the dice score

printf("%s: I'm playing my dice\n", name);

dice = (int)time(&ss) % 10 + 1;

printf("%s: I got %d points\n\n", name, dice);

// Updating the old score

int old = oldScore;

oldScore = old + dice;

// Writing the new score at the same file offset

fd = open("sharedFile.bin", O_WRONLY);

if (playerId == 1) // PLAYER1

{
```

```
            lseek(fd, 0, SEEK_SET);
            write(fd, &oldScore, sizeof(int));
        }
        else if (playerId == 2) // PLAYER3
        {
            lseek(fd, sizeof(int), SEEK_SET);
            write(fd, &oldScore, sizeof(int));
        }
        else // PLAYER2
        {
            lseek(fd, 2 * sizeof(int), SEEK_SET);
            write(fd, &oldScore, sizeof(int));
        }
        close(fd);
        // Sleeping for 2 seconds and signaling the referee before pausing
        sleep(2);
        kill(getppid(), SIGUSR1);
    }
}

void checkWinner(int fd, char *name)
{
    int currentScore;
    // reading the new totals from sharedFile.bin
    read(fd, &currentScore, sizeof(int));
    // printing player's name and total points so far
    if (strcmp(name, "PLAYER1") == 0)
        printf("Referee: PLAYER1's current score: ");
    else if (strcmp(name, "PLAYER2") == 0)
        printf("Referee: PLAYER2's current score: ");
    else
        printf("Referee: PLAYER3's current score: ");
    printf("%d\n", currentScore);
    sleep(2);
    // checking if there's a winner and terminating all processes in case there is
    if (currentScore >= max_size)
    {
        printf("Referee: %s won the game\n", name);
        kill(0, SIGTERM);
    }
}
```

SNAPSHOTS OF OUTPUT



```
user@user-VirtualBox:~/Desktop/Project$ ./a.out proj.c
DiceGame: a 3-players game with a referee

Enter the Maximum points
10


Referee: PLAYER1's current score: 0
Referee: PLAYER1 plays

PLAYER1: my current score is: 0
PLAYER1: I'm playing my dice
PLAYER1: I got 7 points

Referee: PLAYER2's current score: 0
Referee: PLAYER2 plays

PLAYER2: my current score is: 0
PLAYER2: I'm playing my dice
PLAYER2: I got 2 points

Referee: PLAYER3's current score: 0
Referee: PLAYER3 plays

PLAYER3: my current score is: 0
PLAYER3: I'm playing my dice
PLAYER3: I got 6 points
```



```

Referee: PLAYER1's current score: 7
Referee: PLAYER1 plays

PLAYER1: my current score is: 7
PLAYER1: I'm playing my dice
PLAYER1: I got 10 points

Referee: PLAYER2's current score: 2
Referee: PLAYER2 plays

PLAYER2: my current score is: 2
PLAYER2: I'm playing my dice
PLAYER2: I got 4 points

Referee: PLAYER3's current score: 6
Referee: PLAYER3 plays

PLAYER3: my current score is: 6
PLAYER3: I'm playing my dice
PLAYER3: I got 8 points

Referee: PLAYER1's current score: 17
Referee: PLAYER1 won the game
Terminated
user@user-VirtualBox:~/Desktop/Project$
```